第12回

論理回路のプロパティ検証技術(4)

――時相論理による仕様記述法

藤田昌宏

ハードウェアの仕様(プロパティ)の記述に利用される時相論理には，いくつかの種類がある．ここでは，代表的な時相論理であるLTTL(Linear Time Temporal Logic)とCTL(Computation Tree Logic)について解説する．前者は各時刻で各変数の値が一つに決まる時相論理，後者は次の時刻が複数ある時刻列を扱う時相論理である．また，タイミング・チャートの信号は正規表現や正規言語で記述するとコンパクトに表現できる．正規文法と時相論理を組み合わせた拡張時相論理についても紹介する．(編集部)

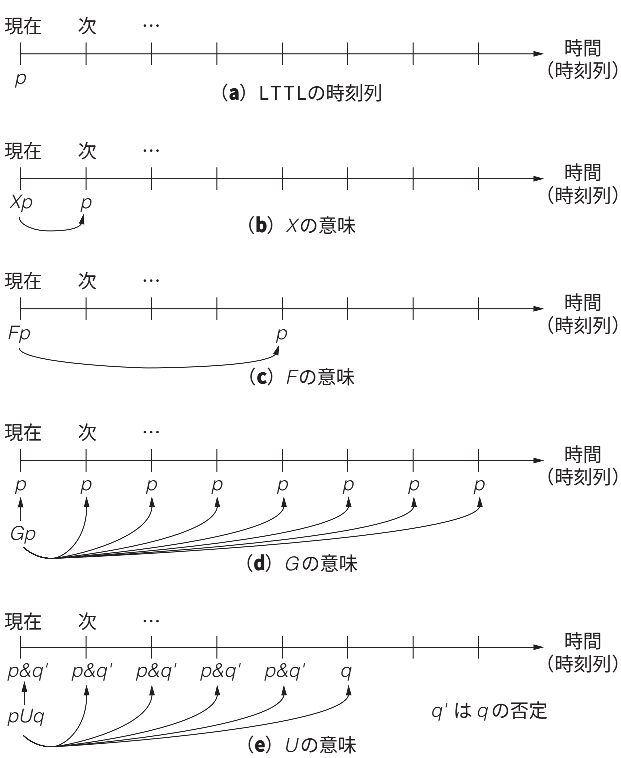


図1　時相論理(LTTL)演算子の意味

時相論理では，つねに現在から見た将来の信号値の取りかたを指定する．LTTL(Linear Time Temporal Logic)では，現在に対する次，その次，さらにその次，…，が一つだけ存在するモデルであり，時刻列が(a)のように一直線になる．(b)のように次の時刻に信号*p*が'1'の場合には，現在，*Xp*が成立する．また，*Fp*は，(c)のように，現在も含めて将来の少なくとも一つの時刻で信号*p*が'1'になる．*Gp*は，(d)のように現在も含めてすべての時刻で信号*p*が'1'となる．*pUq*は，(e)のように，信号*q*が'1'になるまでは，現在も含めてつねに信号*p*が'1'でなければならない．

ここまで，時相論理演算子として，*X*(next，次に)，*F*(sometime，いつかは)，*G*(always，いつも)，*U*(until，～になるまでは)の四つの演算子があることを説明しました．実は，時相論理にもいくつかの種類があります．

今回はこれらを整理しながら，簡単に比較するとともに，タイミング・チャートやハードウェアの動作を記述するうえでの利点や欠点について説明したいと思います．実際には，これらの時相論理のいくつかをまとめたような形で，プロパティを記述するための言語が提案されています．

● 各時刻で各変数の値が一つに決まるLTTL

上述のように，時相論理にはいくつかの種類があります．今まで説明してきたものは，一般にLTTL(Linear Time Temporal Logic)と呼ばれるものです．LTTLでは，**図1(a)**に示すように時間軸上の現在，次，その次というように，現在から無限に続く時刻列があり，その時刻列上の各時刻で，各変数の値が決定されることになっています．

図1(b)～(e)に，各時相演算子*X*，*F*，*G*，*U*の意味を図で表現しました．直感的には，これで意味がわかると思います．*X*は「次」を表し，*Xp*は次の時刻に信号*p*が'1'になることを示しています．また，*F*は「いつか」を表し，*Fp*はいつかは信号*p*が'1'になること(ならなければならないということ．以下同じ)を表現しています．*G*は「いつも」に対応し，*Gp*で(現在時刻以降の)どの時刻でも信号*p*が'1'になることを表現しています．

演算子*F*と*G*は互いに補の関係にあり，片方からもう一

方を定義可能です．直感的に言って，いつも信号pが‘1’であるということは，いつかは信号pが‘0’になるということはない，ということと同じです．このことから，$Gp=(Fp')'$であると言えます（ ’は否定を表す）．また，同様に，$Fp=(Gp')'$であるとも言えます．

最後のUはuntilを表し，少し複雑です．pUqで信号qが‘1’になるまでは，ずっと信号pは‘1’であり続けなければならないということを表現しています．ここで，もしも現在，信号qが‘1’であれば，信号pの値に制限はありません．しかし，現在，信号qの値が‘0’の場合，現在の信号pの値は‘1’でなければならないことを示しています．そのため，$pU(false)$という時相論理式（ただし，$false$はつねに値が‘0’の信号であるとする）は，ずっと信号pが‘1’であり続けなければならないことになるので，Gpと同じ意味（つまり等価）です．ここは少しややこしいので，自分でちょっと考えてみてください．

以上のことから，実はX演算子とU演算子があれば，FとGは必要ないということになります．しかし，時相論理で仕様を記述する際には，F演算子やG演算子を使ったほうがはるかにわかりやすいので，XとUだけでなく，FやGも一般的に利用されています．

さて，ここで重要なことは，各時相演算子は，現在から見た時刻に関する変数の値を規定している点です．また，時相演算子の付いていない変数は，その変数の「現在」の値を示しています．さらに，時相演算子は，いく重にもネストすることもあります．

● タイミング・チャートの関係を時相論理で記述する

一般にタイミング・チャートで表現されているような時間順序関係を時相論理で記述することを考えます．まず基本的な関係として，**図2**に示すような関係，ことばで言うと，「今，信号pが‘1’ならば，次の時刻に信号qが‘1’になる」を考えます．これをもっともすなおにLTTLで表現すると，以下のようになると思います．

p -> Xq

ここで，「->」はimplyを表します．a -> bは$a'+b$として定義され，「aでないか，またはbである」という意味になります．なお，この定義は一般的なもので，論理では，「aならばbである」と言ったときに，aが成り立たなければ何が起こってもよい（何も起こらなくてもよい）という意味になります．

さて，このp -> Xqによって**図2**のタイミング・チャートの関係をきちんと表現していると言えるのでしょうか？確かに，信号pが‘1’であれば，次の時刻に信号qが‘1’でなければならないと規定されています．しかし，上で述べたように，時相演算子のない変数については，現在の時刻の値のみを規定しているので，p -> Xqの中のpは，現在のpの値のみをチェックすることになります．

一般に，**図2**のようなタイミング・チャートで表現された時間順序関係は，現在の時刻についてのみ言っているのではなく，一定の条件下のすべての時刻の規則となっているはずです．そのため，このままではタイミング・チャートで表現されているものと合わないことになります．

信号pが‘1’であるか否かのチェックは，ずっと行わないといけないということから，式全体に「いつも」を表す演算子Gを付ける必要があります．p -> Xqの全体にG演算子を付けて以下のようにします．

$G(p$ -> $Xq)$

これで，pが‘1’になったときの信号qの動作が規定されたことになります．しかし，以下の2点については，動作が規定されていません．

1）信号pが‘0’のとき，信号qがどのようなふるまいをするべきか？

2）信号qが‘1’になるのは（立ち上がるのは），信号pが‘1’になった次の時刻に限られるのか？

厳密に言うと，これ以外にも規定されていないことはあるのですが，ここではこの2点についてのみ考えます．

まず，**1)**については，**図2**のタイミング・チャートをどのように解釈するのかに依存する話になるのですが（したがって，タイミング・チャートにあいまい性があるということになる），とくに規定されていないということであれ

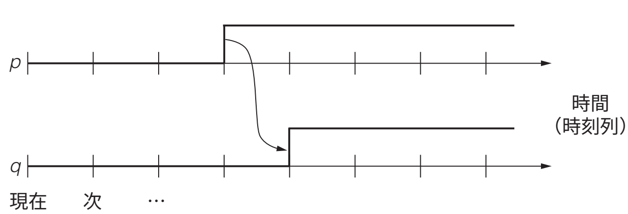


図2　信号pが立ち上がると，次の時刻に信号qが立ち上がるタイミング・チャートの例

一般にタイミング・チャートでは，この図のように信号値間に関する因果関係を矢印を使って示すことが多い．ただし，各矢印が正確にどのような因果関係を示しているのかはかならずしも明確ではない．この例の場合，信号qが立ち上がるのは，信号pが立ち上がったときに限るのか，それ以外にもありえるのかは示されていない．

ばこのままでよいことになります．一方，信号pが‘0’の場合には，次の時刻に信号qが‘0’になる（立ち下がる）必要があれば，G(p’-> Xq’)という条件が付け加わることになります．つまり，以下のようになります．

G(p -> Xq) & G(p’-> Xq’)

これは，信号pが信号qの値を制御していることを表現しています．

2) もタイミング・チャートの理解のしかたに依存する話なのですが，ここでは信号qが‘1’になるのが（立ち上がるのが），信号pが‘1’になった次の時刻に限られる場合と，そうでない場合に分けて考えてみます．

まずは信号qが‘1’になるのが信号pが‘1’になった次の時刻に限られる場合ですが，このケースでは，それ以外の場合に信号qが‘0’から‘1’に変化してはいけないことになります．このことを表現するには，通常，U演算子を用いて，次のように表現します．

G(q’->(q’Up))

これは，信号qの値がいったん‘0’になると，信号pが‘1’になるまで，ずっと‘0’のままであり続けなければならないということを示しています．言い換えると，信号qの値が‘0’から‘1’に変化してもよいのは，信号pが‘1’になった場合だけであるということです．この場合，信号pが‘1’になった「次」の時刻に信号qが‘1’になるということなので，U演算子の左側の時刻を修正して，以下の式が正しく条件を示していることになります．

G(q’->(Xq’)Up)

U演算子の左がXq’となっているのは，信号pが‘1’になったその時刻ではなく，次の時刻で信号qが‘1’に変化するからです．したがって，「信号pが‘1’になったら，またそのときに限って，次の時刻に信号qが‘1’に変化する」というのは，次のようになります．

G(p -> Xq) & G(q’->(Xq’)Up)

このように，二つの時相論理式がペアになって動作を規定することになります．

さて，**2)** において，信号qが‘1’に変換する条件が信号pが‘1’になる場合だけとは限らないケースは，どのように記述するのでしょうか？ 一般的には，G(q’->(Xq’)Up)の中のU演算子の右側の条件が成立しない限り，信号qは‘0’のままでなければならないということですから，このU演算子の右側に信号qが‘1’に変化してよい条件を列挙することになります．例えば，信号pだけでなく信号rが‘1’になっても次の時刻に信号qが‘1’に変化するということであれば，次のようになります．

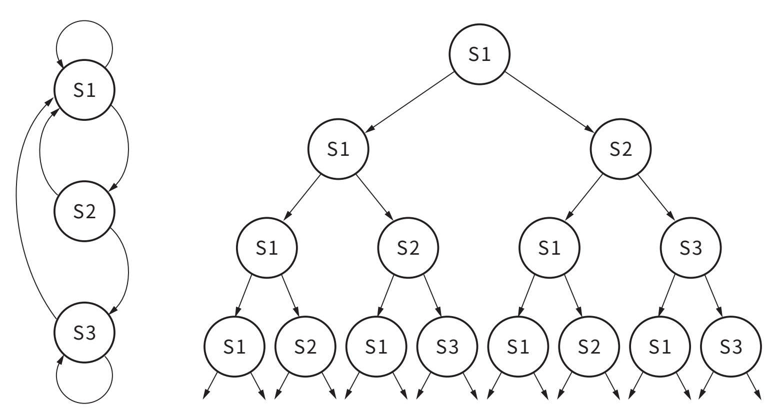
G(q’->(Xq’)U(p＋r))

少し，話がややこしくなりましたが，ゆっくり考えてみてください．タイミング・チャートでは何となく図で表現して時間順序関係を示せていると考えがちですが，厳密な議論を行うと，個々の場合について細かく指定する必要があるのです．時相論理などの論理を使ってハードウェアの仕様を記述する利点は，考慮すべき場合にもれがないかどうかを比較的系統立てて管理できる点です．言い換えると，かなりめんどくさい方法であるとも言えます．

最後にもう一つ．信号pが‘1’になったときに，どのタイミングで信号qが‘1’になるべきかが明確に規定されておらず，いつかは信号qが‘1’になればよい，という仕様（プロパティ）もよく使われます．このような場合，以下のようにF演算子を利用して，「いつかは」という性質を表現します．

G(p -> XFq)

単にFではなくXFとなっているのは，次の時刻以降の



（a）状態遷移表現　（b）計算木

図3
CTL（Computation Tree Logic）
計算木とは，状態遷移図において，初期状態（あるいは現在の状態）から始め，可能な状態遷移列をすべて列挙し，木状にしたものである．（a）の初期状態S1から展開すると，計算木として（b）が得られる．この計算木は無限に続くものである．

「いつかは」ということを表現するためです.

● 次の時刻が複数ある時刻列を扱うCTL

LTTLでは，次の時刻はかならず一つでした．時刻列も**図1**のように1次元の直線として表現できます．これも一つの考えかたですが，次の時刻が条件によっていろいろありうると考えるのも自然です．

時相論理は，もともと哲学者がその考えをあいまい性なく表現して議論するための道具としての側面がありました．そのため，考えかたとして，次がかならず一つに限定されるか，いろいろありうるかというのは，どちらも定義可能です．一般に，次がかならず一つに限定されるものを“直線時間(linear time)の”時相論理と呼び，次が複数あると考えるものを“分岐時間(branching time)の”時相論理と呼びます．

この分岐時間の時相論理の中でもっとも広く利用されているのが，CTL(Computation Tree Logic)と呼ばれるものです．実際，現在製品化されているモデル・チェッキング・ツールのほとんどはCTLに対応しています．

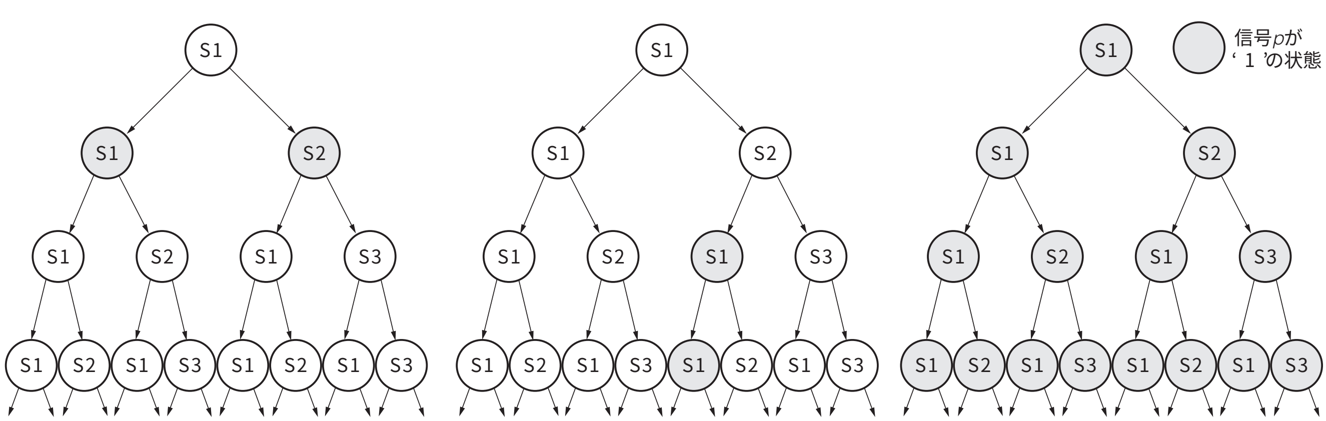
CTLでは，**図3(a)**の状態遷移表現を現在の状態から(図では，初期状態のS1から)順に，可能な状態遷移をすべてたどるように展開していきます．すると，**図3(b)**のように現在の状態を基点(グラフの根)とする木状のグラフが生成されます．これを計算木(Computation Tree)と呼びます．この計算木は，**図2**のLTTLの時刻列の直線とは対照的に，次の時刻が複数ある時刻列であると考えられます．このような次の時刻が複数ある時刻列上で，各信号の値を規定するものが分岐時間の時相論理であり，この場合，CTLと呼ばれます．

● 四つの時相演算子と二つのパス演算子を組み合わせる

CTLにも，時相演算子として*X*，*F*，*G*，*U*があり，その定義はLTTLのときと同じです．CTLではさらに，複数の次の時刻に対応する時刻列があり，それらの上で各時刻の信号に関する値の指定を行うため，パス演算子と呼ばれるものがあります．パス演算子には*A*と*E*があり，*A*はすべての時刻列に対して成立することを要求し，*E*は少なくとも一つの時刻列に対して成立することを要求します．*A*や*E*のパス演算子は，*X*，*F*，*G*，*U*の一つとペアで利用されます．つまり，時相演算子として*AX*，*EX*，*AF*，*EF*，*AG*，*EG*，*AU*，*EU*の8種類の時相演算子があると考えることもできます．このうち，*AX*，*EF*，*AG*の意味を計算木上で説明したものを**図4**に示します．

まず，*AXp*は，すべての次の時刻で信号*p*が‘1’であることを要求します．したがって，**図4(a)**でもすべての次の状態で信号*p*が‘1’となっています．これが，*EXp*の場合には少なくとも一つの次の状態で信号*p*が‘1’となっていればよいということになります．

*EFp*は，少なくとも一つの時刻列に対して，いつかは信号*p*が‘1’になることを要求します．これは，計算木上では，現在から考えて，木全体として少なくとも一つの状態で信号*p*が‘1’になればよいということになり，結果とし



(a) AXの意味　(b) EFの意味　(c) AGの意味

図4　CTL演算子の意味

CTLは，計算木上に定義されたものである．*AXp*は，(a)のように現在の次のすべての状態で信号*p*が‘1’であることを示している．また，*EFp*は，(b)のように現在から見て信号*p*が‘1’となる状態が，いずれかの状態遷移列に少なくとも一つ存在することを示している．これは，計算木全体の中で少なくとも一つはそのような状態が存在することと同じである．*AGp*は，逆にすべての状態遷移列のすべての状態で信号*p*が‘1’であることを示している．すなわち，(c)のように計算木上の状態のすべてで信号*p*が‘1’である必要がある．

て**図4(b)**に示すようになればよいということです．これに対して*AFp*は，すべての時刻列のどれを見ても少なくとも一つの時刻(状態)で信号*p*が‘1’である必要があります．

*AGp*はすべての時刻列に対して，すべての時刻で信号*p*が‘1’であることを要求します．計算木では，現在から見て，すべての状態で信号*p*が‘1’である必要があるということになります(**図4(c)**)．

● CTLでは式の長さに比例する手間で検証が終わる

このように，CTLなどの分岐時間の時相論理は少しややこしいと感じるかもしれませんが，パス演算子の*A*と*E*を利用するメリットもあります．それは，任意のCTL式に対して，状態遷移図上の各状態がその式を満たすか満たさないかで，互いに補な関係になるようにきれいに分かれることです．つまり，状態全体の集合から式を満たす状態の集合を取り去ると，式を満たさない状態の集合と一致する(逆も成立)ということです．このため，式を満たす集合を求めれば，式を満たさない状態の集合も自動的にわかります．

この性質があるため，CTLの場合，与えられた式を満たすか否かのチェック(モデル・チェッキング処理)を部分式から順にボトムアップに調べていくことで，式全体を満たすか否かのチェックが行えることになります．つまり，仕様の式の長さに比例する手間でモデル・チェッキングが可能となるわけです．このことは，1982年に米国Carnegie Mellon UniversityのClarke教授らのグループによって初めて示され，その後，CTLによるモデル・チェッキングが広がりました．

LTTLの場合には，上記のような性質は成り立ちません．LTTLでは，すべて*A*パス演算子が付いている時相演算子のみ用意されていると考えることができます．したがって，前回までに示したように，LTTL式をいったん状態遷移表現に展開して検証することになります．このLTTL式から展開して生成された状態遷移表現中の状態数は，時相論理式の長さに対して，最悪，指数関数的に増大する可能性があります．その意味で効率的ではありません．

ただし，実際には，仕様として記述する時相論理式についてそれほど複雑なものを記述することはないため，LTTLよりCTLが大幅に有利ということにはなっていないようです．ハードウェア記述言語から自動変換された時相論理式など，特別な場合には，式が膨大になることがありますが，そのような場合以外は，LTTLとCTLの処理効率にはそれほど差はありません．

LTTLとCTLでは，実は「表現できるもの」が異なるのですが，実際の設計に対する仕様記述ではその差が意識されることはあまりありません．現実問題として，ほとんど同じであると理解しても作業上問題ありません．ただし，CTLでは，*A*パス演算子と*E*パス演算子を使い分ける必要があり，その意味では若干の慣れが必要であることは事実です．

最後に一つだけ．LTTLやCTLで，直接表現できない仕様の例を紹介します．今，信号*p*の値が，現在から見て偶数時刻では‘1’になる，という仕様を考えます．奇数時刻については条件はなく，‘0’でも‘1’でもよいということです．この仕様はLTTLで直接表現することはできません．*p* & *G*(*p* -> *Xp*’) & *G*(*p*’-> *Xp*)とすると確かに偶数時刻では信号値は‘1’になりますが，奇数時刻ではかならず信号値が‘0’となり，仕様と合いません．

実はこのような仕様はLTTLやCTLでは決して表現できないことを証明できます．また，ちょっとした拡張で表現できるようにもなります．少し考えてみてください．

● 信号値の変化は正規表現や正規言語で記述できる

時相論理と並んでハードウェアの仕様記述，とくに信号の値の時系列を指定する手法として，正規表現(regular expression)や正規言語(regular language)，正規文法(regular grammar)を利用する方法があります．これらの三つは基本的に同じものであり，式で表現するか，言語で表現するか，生成規則を表す文法で表現するかが異なるだけです．取りうる値の組み合わせに関して，可能な値の列挙，特定のシーケンスの繰り返しなどを利用することで，信号の値を指定する手法です．信号値が‘0’か‘1’の2値であるとすると，“010101…”のような‘0’と‘1’の無限の繰り返しは，一例として，以下のように表現できます．

(01)*

また，信号値が‘0’か‘1’のどちらでもよい場合には，以下のように表現できます．

0＋1

これらを組み合わせると，例えば現在から見て偶数時刻では信号値が‘1’，奇数時刻では‘0’でも‘1’でもよいという仕様は，次のように表現できることになります．

(1(0＋1))*

つまり，信号値‘1’の次は‘0’でも‘1’でもよいというこ

とが無限に繰り返されるということになります．信号pがこの仕様を満たすことは，例えば次のように表現します．

$p = (1(0+1))^*$

この例は，**図5**に示すように上述のLTTLやCTLでそのままでは表現できないとされていた仕様であることに注意してください．

このように，時相論理と正規表現で表現できる仕様の範囲は異なります．細かい議論はかなり複雑なのでここでは示しませんが，一つ言えることは，時相論理や正規表現をそのまま利用することは少なく，実際にはこれらをもとに作られた仕様記述言語を利用します．この仕様記述言語では，さまざまなくふうが施してあり，仕様の表現に関して実用上問題がないようになっています．

さて，上の例を正規文法で表現すると次のようになります．

A → 1B
B → (0＋1) A

最初にAから始めて，上記の規則を順次適用していきます．すると，まず信号値‘1’が生成され，Bという状態になります．次にBの規則を見ると信号値として‘0’か‘1’のどちらでもよいことがわかります．次の状態はAなので，以上の値が繰り返されることになります．

なお，値を列挙することで，可能な値を示すこともできます．例えば2時刻にわたる信号値が“00”か“11”のどちらかであることを示すには，以下のようになります．

(00＋11)

したがって，信号値として“00”か“11”を繰り返す仕様(例えば，“001100110000…”という仕様)は，次のように表現できます．

$(00+11)^*$

● 正規表現では信号値の因果関係がわかりにくい

このように正規表現は信号値の時系列を柔軟に指定することができ，ハードウェアの仕様記述法としては簡潔で便利な手法です．ただし，タイミング・チャートで表現されるような信号値の因果関係を表現するには，正規表現だけではわかりにくくなることもあります．

例えば，先ほどの「信号pが‘1’になると，次の時刻に信号qが‘1’になる」というのは，信号pとqの値の時系列をペアで同時に指定することになります．信号pとqに許される値の組み合わせを正規表現で列挙するのです．今，信号pとqの同時刻の値のペアを

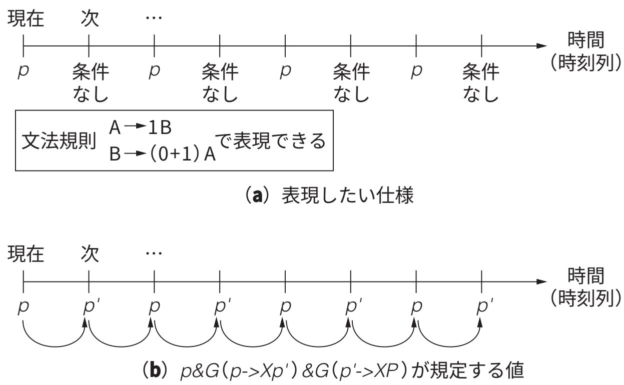
$[p, q]$

のように表すとすると，正規文法では下記のように示せます．

A → [0,0] A
A → [0,0] B
B → [1,0] C
C → [?,1] D
D → [?,?] D

ここで，?は値として‘0’でも‘1’でもよいことを示すものとします．最初の一定期間，信号pとqの値が‘0’である期間があり(状態A)，その後，信号pが‘1’となり(状態B)，その次に信号qが‘1’となり(状態C)，その後は信号p，qとも決まった値を指定されない(状態D)というように表現されています．

しかし，このようにすると，信号間の因果関係が明確でないという問題があります．つまり，信号qの値が‘1’に立ち上がる原因が明確に表現されておらず，ハードウェアの仕様記述としては不明瞭であると言えます．また，信号値の間の因果関係は，時相論理のような論理式で記述するほうがはっきりし，わかりやすいと言えます．そして，時相論理と正規表現を組み合わせるのがもっとも良いと考えられます．この考えかたに基づいて提案されたのが，拡張時相論理(Extended Temporal Logic)と呼ばれるものです．



(a) 表現したい仕様

(b) p&G(p->Xp')&G(p'->XP)が規定する値

図5　現在から見て偶数時刻で信号pが‘1’であるという仕様の表現

現在を時刻0としたとき，偶数時刻では信号pが‘1’であり，奇数時刻では信号pの値に対する制限がないという仕様をLTTLで表現することはできない．p & $G(p$ -> $Xp')$ & $G(p'$-> $Xp)$というLTTL式は，確かに偶数時刻で信号pが‘1’であることを示している．しかし，奇数時刻で‘0’となるべきことも示しており，表現したい仕様とは異なる．一方，正規文法などを利用すると，これを簡単に表現できる．

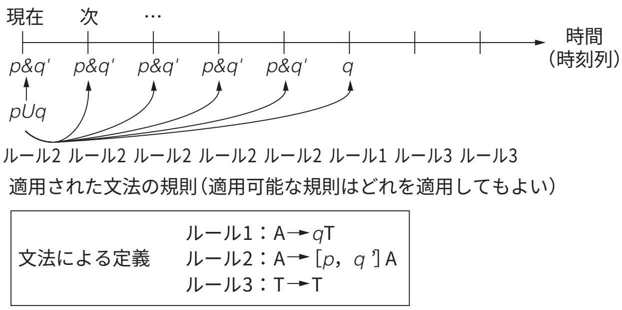


図6　*pUq*の文法規則による表現(時相演算子の定義法)

LTTLやCTLで表現しやすいが，正規表現では煩雑になってしまうもの，あるいは逆に正規表現では簡潔であるが，LTTLやCTLでは表現にかなりくふうが必要なものなど，さまざまな仕様が考えられる．そこで，LTTLやCTLに対して，正規文法を使って自由にカスタム・メイドの時相論理演算子を定義して利用する手法が提案されている．例えば*pUq*は，図に示すような文法で表現できる．三つの規則を適用することにより，*pUq*と同じ動作を表現できている．

● 正規文法と時相演算子を組み合わせて柔軟に記述

拡張時相論理は，実は正規文法でも時相演算子を表現できるという点に着目して考えられた時相論理です．正規文法を利用して，時相演算子を必要に応じてその場で定義して利用できます．例えば，

A → 1A

という規則からは，“111111…”が生成されるので，ずっと‘1’であるということを示しています．したがって，これはG(always)と同じ意味です．否定が定義されていれば，G演算子からF演算子が作れます．また，pUqも以下のように定義可能です．ここでは，p，qで信号p，qが‘1’であること，p'，q'で信号p，qが‘0’であることを示しているとします．

A → qT

A → [p，q'] A

T → T

状態Aから始めると，まず信号qが‘1’なら次の状態はTとなり，いったん状態Tになると無条件でただ繰り返すだけです．また，信号qが‘0’の場合には，信号pが‘1’であることが条件となり，次の状態もAのままで，この動作を繰り返します．結果的にpUqと同じ動作であることがわかります．**図6**に示す説明もいっしょに参考にしながら考えてみてください．

このように，正規文法を使って時相演算子を定義することができるので，それらと通常の時相論理演算子を自由に組み合わせて仕様を記述することが可能です．

理屈上は，もっとも強力かつ柔軟に記述できる時相論理であると言えます．しかし，実際の設計では，複雑な仕様を表現することはあまりないので，それほど普及していません．しかし，ビヘイビア合成などが普及し始めると，クロック単位の動作が未決定なものの仕様を明確に表現しておきたいという要求が高まることが考えられます．そうなると，拡張時相論理のようなものの利用が進む可能性があります．

● 次回は仕様記述言語について解説

以上，時相論理や正規表現の基本について説明してきました．ここまでの議論はかなり数学的な話でしたが，実際の設計の仕様記述を行うには，計算機処理が可能な言語として定義する必要があります．そのような言語を仕様記述言語と呼びます．

次回は，この仕様記述言語や関連技術について説明してモデル・チェッキングの説明を終えたいと思います．また，今回紹介しきれなかった時相論理として，インターバル時相論理(Interval Temporal Logic)があります．柔軟な仕様記述，あるいはビヘイビア記述には便利な時相論理なので，これの考えかたについても可能なら紹介したいと思います．

ふじた・まさひろ
東京大学 大規模集積システム設計教育研究センター（VDEC）教授

<筆者プロフィール>
藤田昌宏．1985年，東京大学大学院情報工学専門課程修了．工学博士．同年，富士通研究所入社．論理設計用CAD技術の研究開発に従事．1993年より米国富士通研究所へ出向，CAD研究グループを立ち上げる．2000年3月より東京大学大学院工学系研究科電子工学専攻教授，2004年4月より東京大学大規模集積システム設計教育センター(VDEC)教授．ディジタル・システム設計支援技術，特にシステム・レベル合成，論理合成，論理検証を中心に研究している．